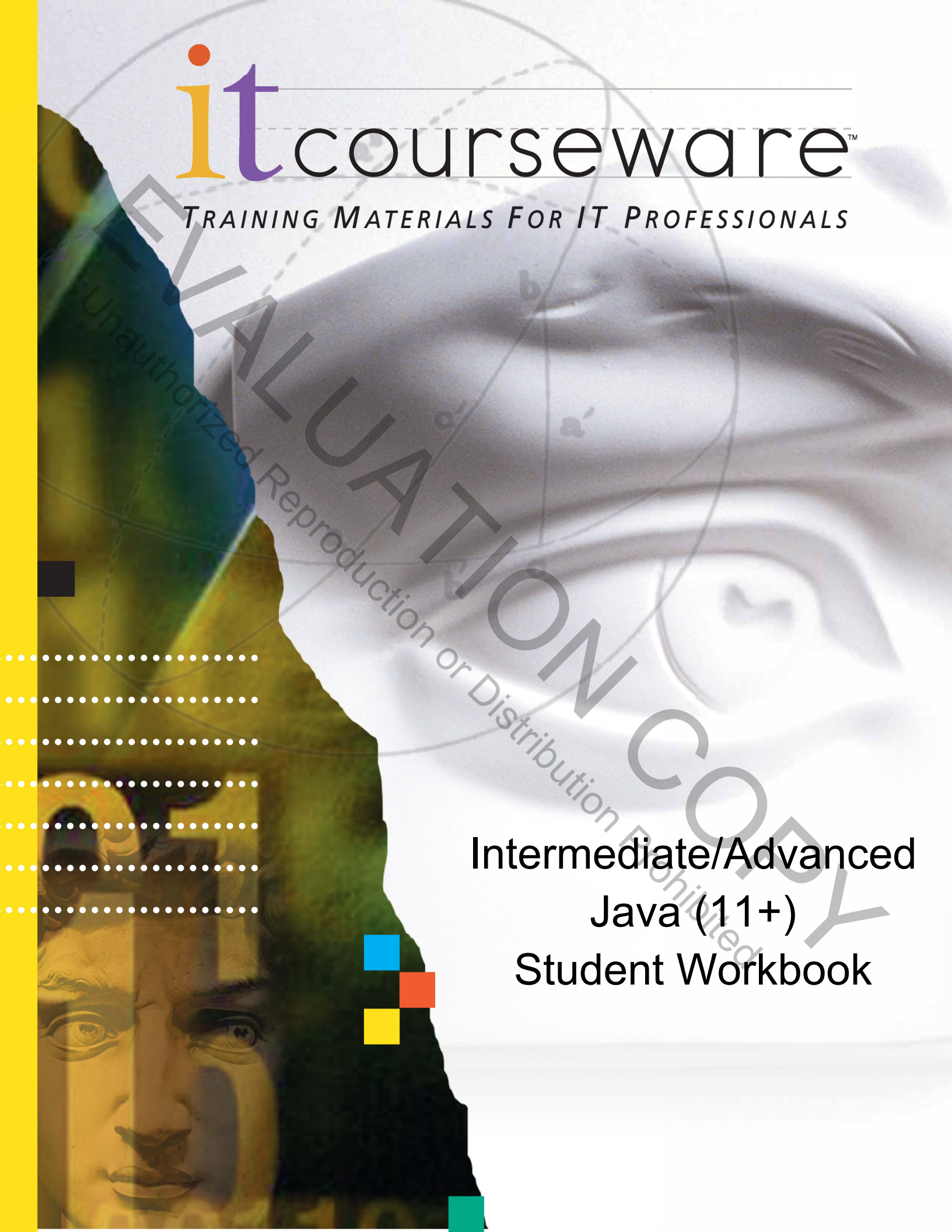


itcourseware™

TRAINING MATERIALS FOR IT PROFESSIONALS

Intermediate/Advanced Java (11+) Student Workbook

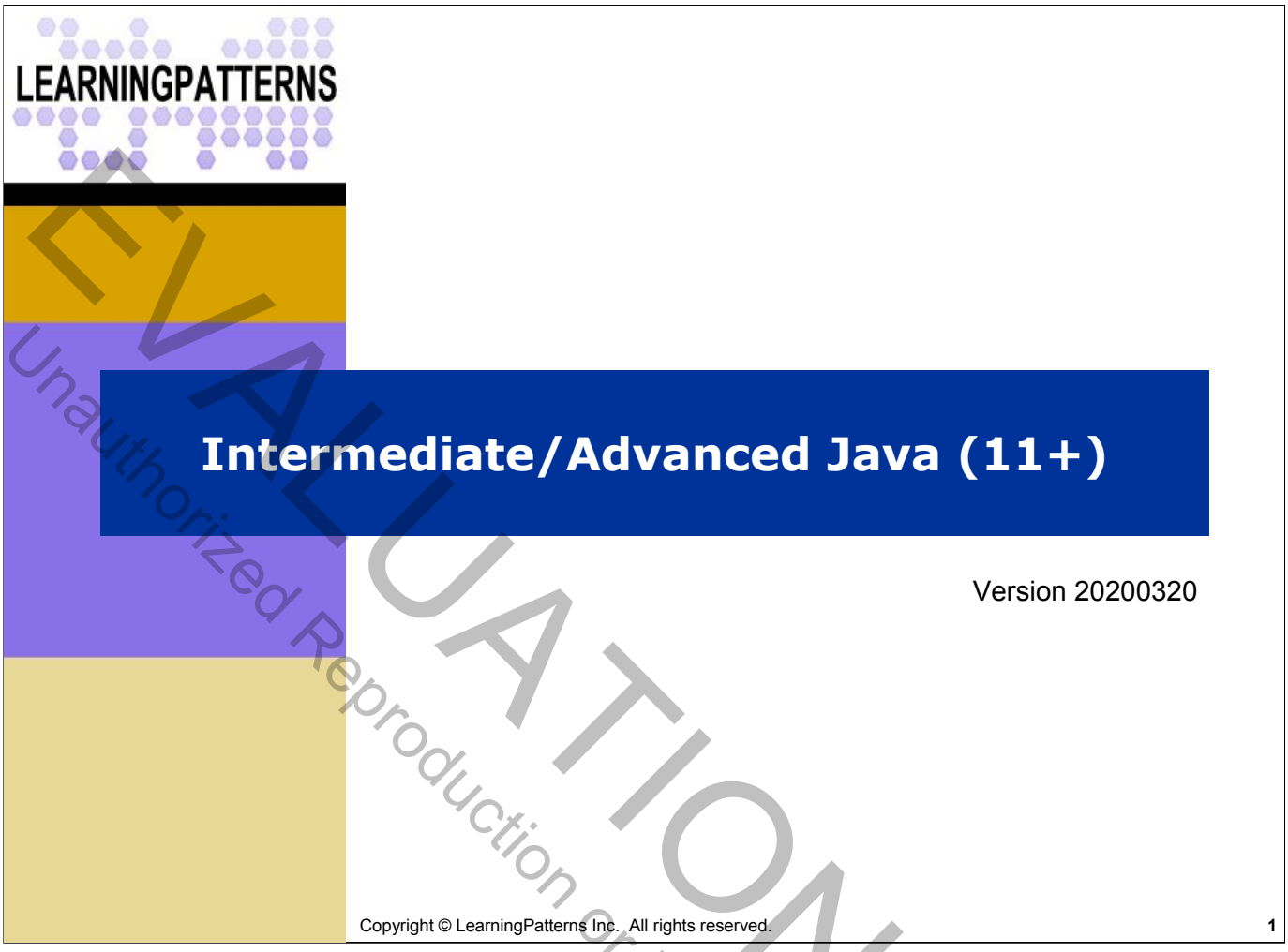
For more information about Java Enterprise Java, or related courseware, please contact us. Our courses are available globally for license, customization and/or purchase.

LearningPatterns. Inc. Services@learningpatterns.com | www.learningpatterns.com

Global Courseware Services 982 Main St. Ste. 4-167| Fishkill, NY 12524 USA
212.487.9064 voice and fax

LearningPatterns

Table of Contents: Intermediate/Advanced Java (11+)



Notes:

- Version 20200320

Course Overview

- 5-day course to deepen your understanding of the Java language, its core APIs, and OO design / development
  - **Goal**: help you become a better Java and OO developer
- **Assumption:** you already know Java (at least the basics)
- It covers the following high-level areas:
  - **UML** and **design patterns**
  - Using **composition and inheritance** effectively
  - Advanced language features: **inner classes**, **writing generic classes**, **custom annotations**, **reflection**
  - Modern language features: **lambda expressions** and **functional programming**, **Stream API**, **Java modules**, **type inference**
  - **Unit Testing**: JUnit used extensively throughout the course

Introduction 2

Notes:

Course Overview

- This is a **very full** intermediate-advanced level course
  - Builds on basic Java knowledge to use it productively
  - Goes well beyond the basics
- Be prepared to work hard and learn a great deal!
- The course contains numerous hands-on labs
  - They exercise all the important concepts discussed
  - Lab solutions for the course are provided
- The course is suitable for **Java 11** and later

Notes:

- You may find the course quite challenging, and that's okay.
  - The primary goal of the course is to help you become a solid Java / OO developer. Java is so big now, a 5-day Intro course + a few months experience just isn't enough anymore.
  - The course pushes you to work hard, and we don't go into every little detail like in an Intro course – we're beyond that at this point.
- There will be a lot of hands-on work in the lab exercises, and you may see some things in the starter code that are new to you.
  - These are things that you will see every day as a Java developer, and you need to start getting comfortable with them.
  - By the end of the course, you will be a much more confident Java developer.

Course Agenda

- Preface: **Java State of the Union**
- Session 1: **Review – Basics**

See **Review Sessions – Guidelines** immediately following this agenda

- Session 2: **Review – Inheritance and Interfaces**
  - Includes a primer on **UML** (used throughout the course)
- Session 3: **JUnit**
  - Starting here, most of the labs will be run as JUnit tests
- Session 4: **Collections and Generics**
- Session 5: **Techniques of Object Creation**
  - Includes an overview of **design patterns**
  - Several patterns are introduced in the course

Introduction 4

Notes:

Course Agenda

- Session 6: **Using Composition and Inheritance Effectively**
- Session 7: **Inner Classes**
- Session 8: **Annotations**
- Session 9: **Reflection**
- Session 10: **Lambda Expressions**
- Session 11: **Streams**
- Session 12: **Introduction to Modules**
- Session 13: **Working with Modules**

By the end of these two sections, you **will** understand Functional Programming in Java

Introduction **5**

Notes:

Review Sessions – Guidelines

- Review sessions 1 and 2 are included for a few reasons:
  - **Reinforce and deepen understanding** of Java core concepts
  - **Provide a level set**, to get ready for the remaining sessions
  - **Introduce features** you may not have had exposure to
    - Even if you've been doing Java for a while
- Content of the review sessions is **two-fold**:
  - Accelerated, yet thorough hands-on review of Java fundamentals
    - Attention given to OO design and implementation principles
  - Coverage of **newer language features** (Java 8 **and later**)
- Spend class time on the most relevant areas in these reviews
  - Perhaps the entire review sessions are covered in full
  - Or you focus more on the new features material

see notes

Introduction 6

Notes:

- This is a **very full** 5-day course, which was designed to be used in a variety of contexts:
  - Developers currently working in Java and want to advance their knowledge.
  - Seasoned OO developers from other languages that want to learn Java at a more advanced level than an intro course would provide.
- This will affect how the course is best delivered – specifically, the content, pace, and labs.
  - Based on student backgrounds and experience level, and interests and goals, the course can be conducted so that more class time is spent in the areas of most value.
  - Spend your time wisely!

Typographic Conventions

- Code in the text uses a fixed-width font

  `Catalog catalog = new CatalogImpl()`
- Code fragments are the same, e.g., `catalog.findById(4)`
- We **bold/color** text for emphasis
- New terms are often introduced in ***bold italics***
  - Filenames and paths are also in italics, e.g., *Catalog.java*
- Notes are indicated with a superscript number (1) or a **star ***
- Longer code examples appear in a separate code box (below)

```
package com.javatunes.catalog;

public class CatalogImpl implements Catalog {
   public MusicItem findById(long id) {
      ...
   }
}
```

Introduction 7

Notes:

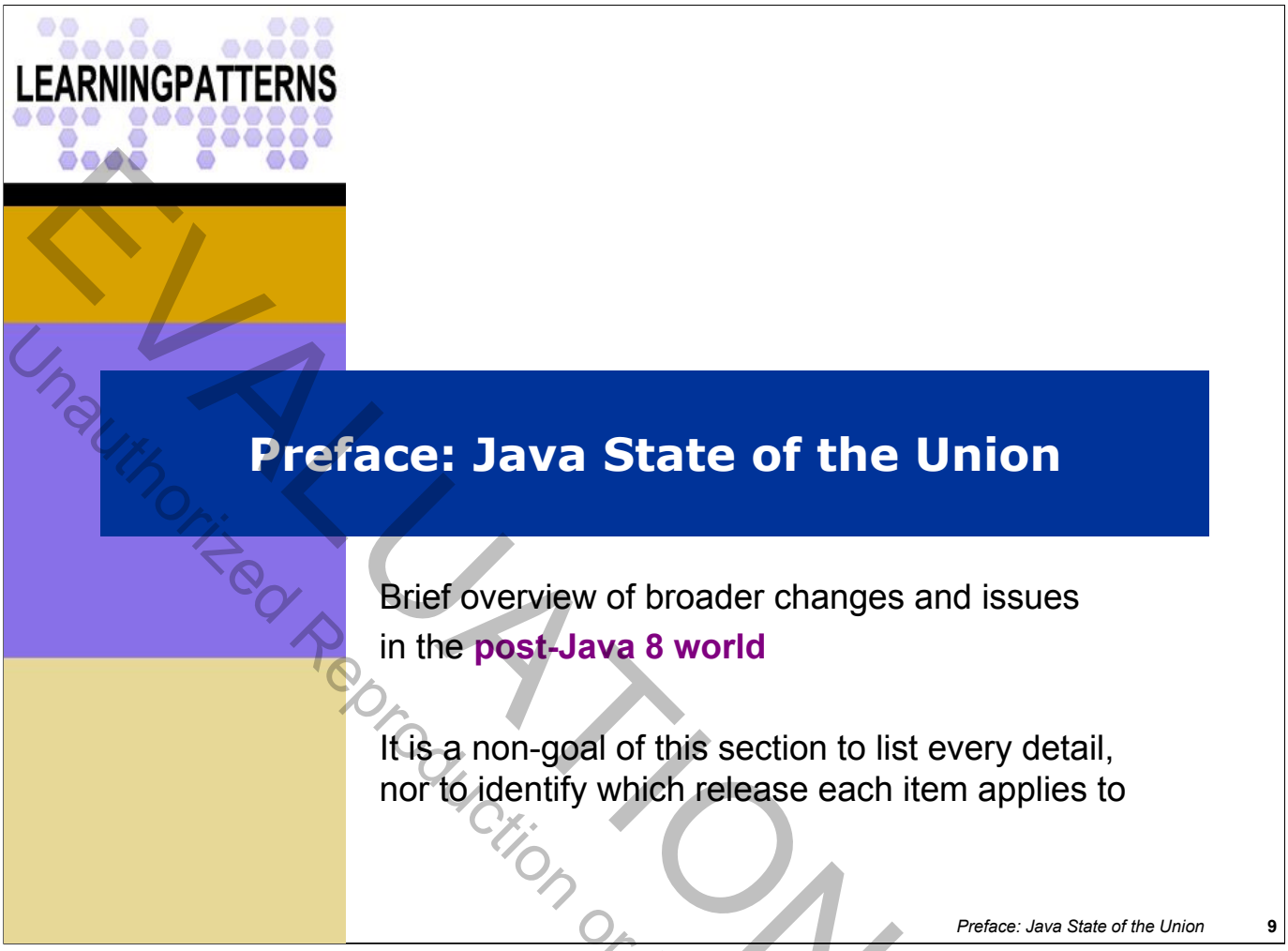
(1) If we had additional information about a particular item in the slide, it would appear here in the notes, with the appropriate superscript.

- We might also include related information that pertains more generally to the slide.

Labs

- The course has numerous hands-on lab exercises
  - Many use types from a fictional case study called **JavaTunes**
    - An online music store
  - The lab instructions are separate from the main manual
- Setup files are provided with skeleton code for the labs
  - Students add code focused on the topic at hand
  - There is a solution zip with completed lab code
- Lab slides have an icon in the upper right corner of the slide
  - The end of a lab is marked with a stop sign

Notes:



Notes:

New Java Release Cycle

- New "major" version every 6 months – **March** and **September**
  - Started with Java 9 in Sep 2017
  - Two kinds of releases, detailed below
- **Feature Release**
  - Implementation enhancements, possible new language features
  - Not intended for long-term use
  - **Superseded** after 6 months, with release of next version
- **Long-Term-Support (LTS) Release**
  - This is what you're accustomed to working with – up to Java 8
  - Plan is to mark a release as LTS every **3 years**
- Details in Oracle Java SE Support Roadmap [see notes]

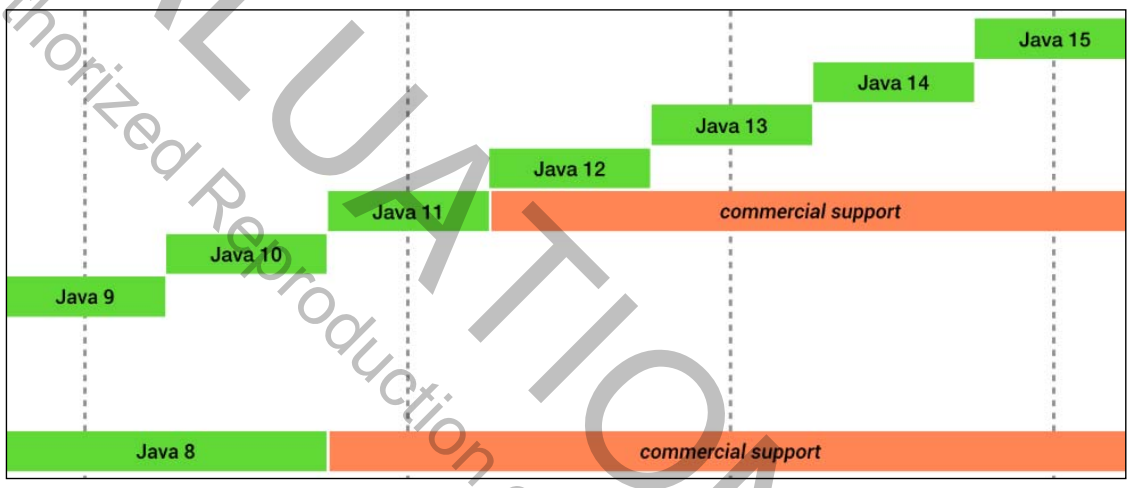
Preface: Java State of the Union **10**

Notes:

- From the **Oracle Java SE Support Roadmap** at *http://oracle.com/technetwork/java/java-se-support-roadmap.html*.
  - For product releases after Java SE 8, Oracle will designate a release, every three years, as a Long-Term-Support (LTS) release. Java SE 11 is an LTS release. For the purposes of Oracle Premier Support, non-LTS releases are considered a cumulative set of implementation enhancements of the most recent LTS release. Once a new feature release is made available, any previous non-LTS release will be considered superseded. For example, Java SE 9 was a non-LTS release and immediately superseded by Java SE 10 (also non-LTS), Java SE 10 in turn is immediately superseded by Java SE 11.
- This is a **very** important document, packed with detail (and myriad footnotes). If you are responsible for maintenance and updates at your organization, you really need to read it in its entirety.

Versions and Support – Illustrated

- Except for LTS releases, these are really just 6-month "technology previews"
  - Official term is "feature release"
- LTS releases span several feature releases
  - **Java 17** is the next planned LTS, in September 2021



Preface: Java State of the Union 11

Notes:

- This release schedule offers some nice benefits, in that updates are more frequent, and predictable. (Though the versioning and non-LTS vs. LTS releases will take some getting used to.)
  - As a consequence, with more frequent releases, it can be more difficult to quickly find the applicable documentation for a specific release. Release Notes and other docs for all releases can be found by simply starting at your JDK installation, and opening *<jdk-install>/README.html*, then click and browse from there.

- Here is a detailed history of releases and updates since Java 9. Look it over closely – see how it works?

- Given the new release schedule, the next LTS release after 11 will be 17 LTS, slated for September 2021.

Non-LTS and LTS releases are more frequent now

– **9.0**	**Sep 2017**	**March** and **September** each year
9.0.1	Oct 2017	one month later, coincides with below
9.0.2	Jan 2018	**Jan-Apr-Jul-Oct** update schedule
– **10.0**	**Mar 2018**	JDK 9 is **dead**
10.0.1	Apr 2018	
10.0.2	Jul 2018	
– **11.0 LTS**	**Sep 2018**	next LTS in **3 years**, JDK 10 is **dead**
11.0.1	Oct 2018	
11.0.2	Jan 2019	– **12.0** **Mar 2019**
11.0.3	Apr 2019	12.0.1 Apr 2019
11.0.4	Jul 2019	12.0.2 Jul 2019
11.0.5	Oct 2019	– **13.0** **Sep 2019**
etc., until next LTS		13.0.1 Oct 2019

What's In

- **Java Modules** *
  - **All code** now resides in a module (of some type)
    - **Including** all your Java 8 code written years ago (more on this later)
  - We now have ***modulepath*** in addition to the classpath
- **JShell** – REPL shell for quick-coding and running scripts
- **HTTP Client** – replaces outdated `HttpURLConnection`
- **Reactive Streams** – non-blocking pub-sub stream processing
- Other enhancements – see notes
  - **Collection factory methods** *
  - **Multi-release JARs** (MR-JARs)
  - **Type inference** – Java has a **`var`** now *
  - **Enhanced `switch`** construct *

JPMS = Java Platform Module System

* covered in the course

Preface: Java State of the Union 12

Notes:

- The Java Platform Module System (JPMS), initially called Project Jigsaw, appeared in Java 9.
  - It is the most significant new feature, and has the potential to affect nearly every aspect of Java development and runtime execution – or not... As we'll see later, even if you "opt out" of creating your own modules, you still need to know how it all works.
- Additional miscellaneous capabilities include **Process API** updates, and a new **Stack-Walking API** for working with exception traces.
- There have also been platform runtime and monitoring changes.
  - G1 is the default collector + new Epsilon GC (no-op), and ZGC.
    - The **Epsilon** garbage collector is a **no-op**, particularly relevant for short-lived programs that don't benefit from the cost of GC. *http://openjdk.java.net/jeps/318*.
    - **ZGC** is a low-latency concurrent collector targeted at very large heap applications. *http://openjdk.java.net/jeps/333*.
  - **Java Flight Recorder** has been donated to open source, no longer requiring a commercial license. *http://openjdk.java.net/jeps/328*.

What's Out

- Java 9 introduced new "attributes" of deprecation
  - `since` and `forRemoval` can now be included in `@Deprecated`
  - And note that items ARE now being marked for actual removal
  - Time to finally take this seriously and change your old code...

```
@Deprecated(since="1.2", forRemoval=true)
```

- Java 11 took this one step further, **physically removing them**
  - Overlapping APIs from Java EE (now Jakarta EE)
  - Client-side UI items
    - **Applets** deprecated in Java 9 and removed in 11
    - **Java Web Start** removed in Java 11, with no replacement
    - **JavaFX** extracted out to OpenJFX – *http://openjfx.io*
- No more JRE
  - Instead, build a (smaller!) custom runtime image with `jlink`

Preface: Java State of the Union 13

Notes:

- The new JDK tool **`jdeprscan`** can be run on your code, and it will tell you all the places you're using deprecated items. (So now you really have no excuse...)
- Several overlapping APIs from Java EE have been physically removed in 11. The most likely to affect you is the removal of **JAXB** and **JAX-WS** – the **`javax.xml.bind`** packages and **`javax.xml.ws`** packages. *http://openjdk.java.net/jeps/320*.
  - If you need them, just provide the libraries (JARs) as for any other external dependency.
- Note: with all the client-side UI pieces being removed or extracted out, Swing and AWT are **still in**.
  - Don't need 'em in your application? See "No more JRE."
- There have also been changes to the Java runtime and monitoring tools included with the JDK.
  - **No more JRE** – instead, build a (smaller!) custom runtime image with **`jlink`**.
  - **Java VisualVM** (`jvisualvm`) and **Java Mission Control** (`jmc`) have been extracted out as separate downloads now, and **Java Flight Recorder** has been donated to open source.
    - This was mostly done to align Oracle JDK with OpenJDK. As of Java 11, these two are now "functionally equivalent, with cosmetic differences."

What We'll Cover

- We can't do everything in 5 days (!)
  - We've chosen the features that are most relevant for this course
- To dive into all the major new features, there is a 2-day ***Java Modules and New Features*** course that includes:
  - Advanced topics in Java modules
  - Multi-release JARs (MR-JARs)
  - HTTP Client
  - JShell
  - Reactive programming in Java
  - Creating custom runtime images with `jlink`

Notes:

Thinking in Java 9+

- JDK command-line tools have evolved over time
  - More options / flags than ever before
  - Many of these can now be expressed in several forms:

```
$ java -cp
$ java -classpath
$ java --class-path     // this is the new form

$ java -p
$ java --module-path    // this is the new form
```

  - **NOTE:** since some of the new options offer only the new form, we will be favoring this one (in the slide examples and labs)

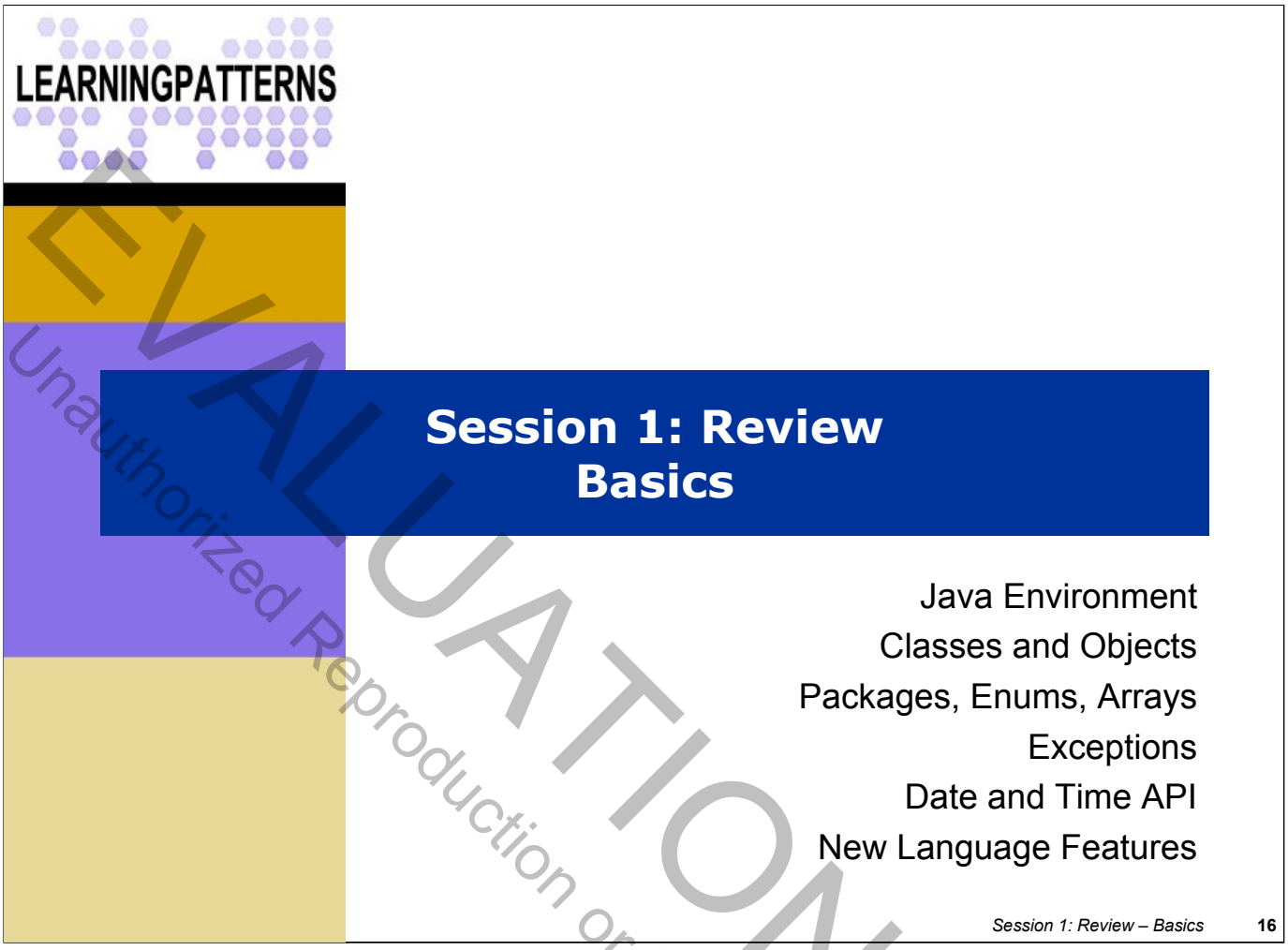
- New compiler option **`--release`**
  - Replaces legacy `-source` and `-target` flags

Preface: Java State of the Union 15

Notes:

- The new `--release` option would be used as: `javac --release <version> ...`
  - Source code is checked for language features not available in <version>.
  - Also checks that you're not using newer APIs not available in <version>.
  - Generates correct bytecode for the target release.

- Previously, most people would just use an older JDK for compilation and testing, and not worry about `-source` and `-target`.
  - The new `--release` option provides comprehensive source code verification and bytecode generation to support older Java runtimes.
  - This means you no longer need to have a previous JDK installed to support earlier releases.



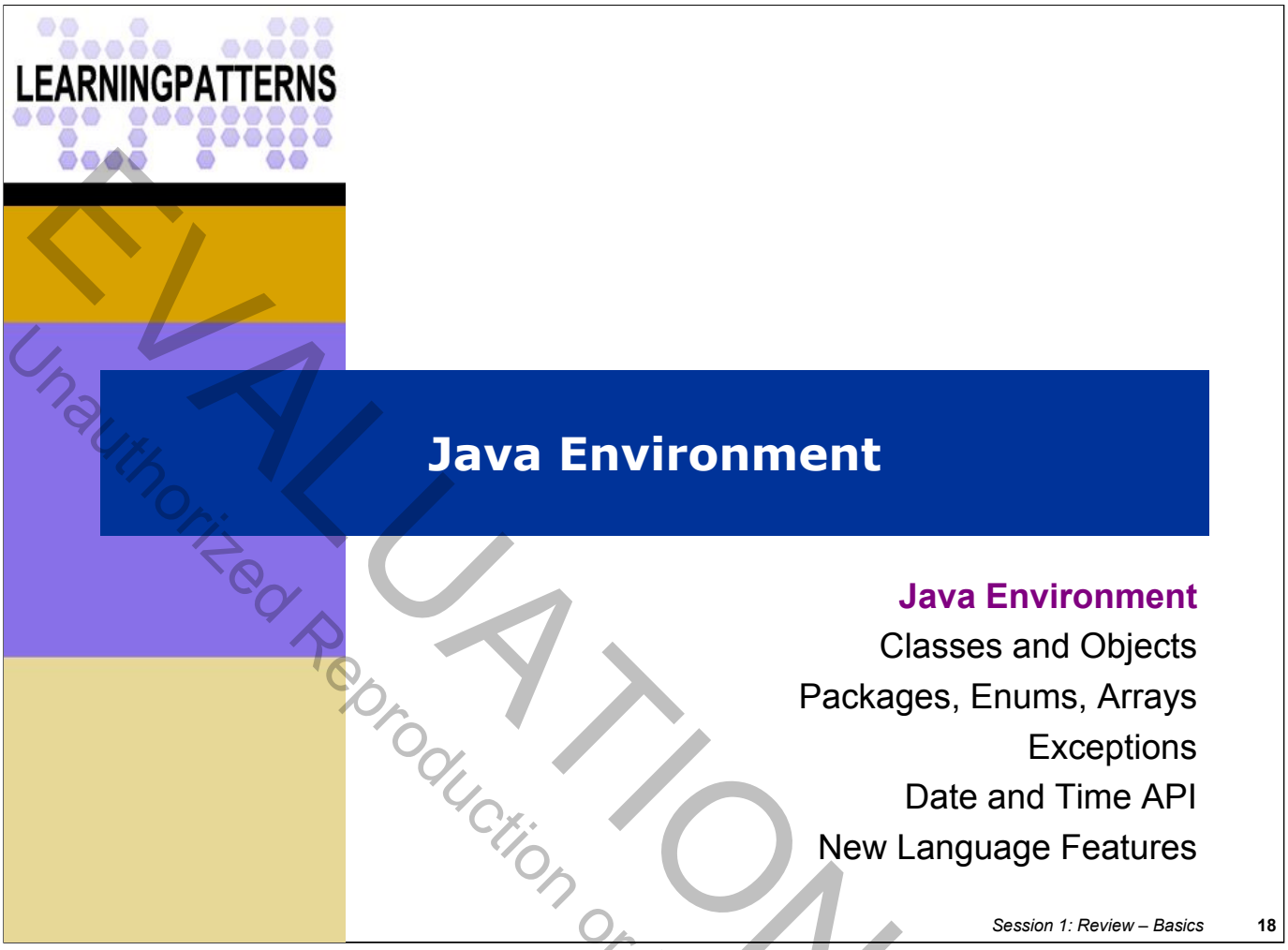
Notes:

Session Objectives

- Review Java landscape, development / runtime environment
- Review basic constructs: classes & objects, packages, enums
  - Build on this: `equals()` and `hashCode()`, `Comparable` and `Comparator` (and their contracts), static fields and methods
  - Reaffirm principles: data encapsulation, cohesion
- Review basic constructs: arrays, flow of control, exceptions
- Discuss important new features in Java 8 and beyond
  - Date/Time API, Java modules, type inference, switch expressions
- **Goal**: warm up with some basic Java and OO review
  - **And start to build on that**
  - This "review" may yield lots of new understanding! (a good thing)

Session 1: Review – Basics 17

Notes:



Notes:

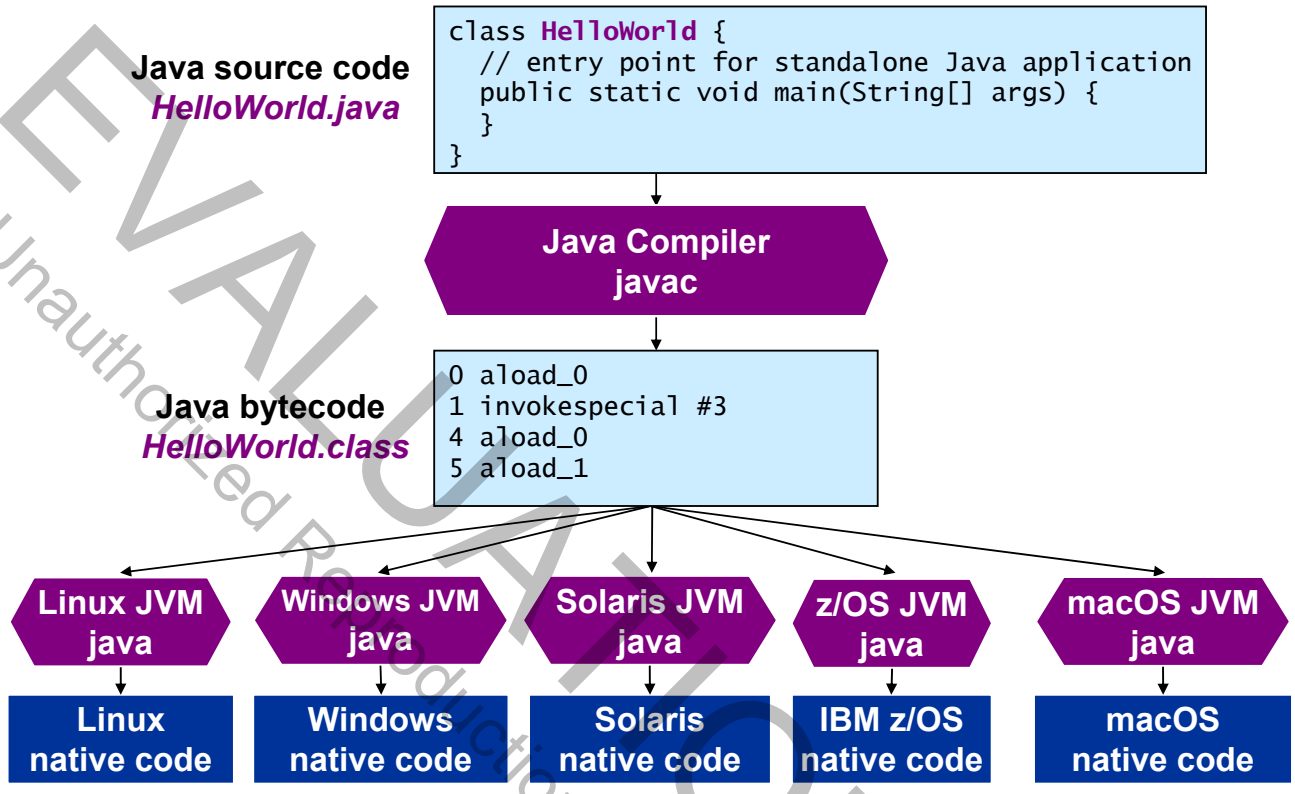
Java – a Language and a Platform

- Source code (*.java*) is compiled into bytecode (*.class*)
  - Generally in an IDE such as Eclipse, IntelliJ, etc.
- Bytecode executes on the Java Virtual Machine (JVM)
  - Which in turn, executes on the target operating system
- Three main platforms:
  - **Java SE** – compiler, runtime (JVM), and standard API libraries
    - `String`, `System`, `JButton`, `Collection` – thousands of classes
    - Plus other utilities – `jar`, `javap`, etc., found in ***JAVA_HOME/bin***
  - **Java EE** – specs and APIs for back end enterprise systems
    - Implementations provided by app servers – JBoss, WebLogic, etc.
    - Transferred to Eclipse Foundation, and renamed **Jakarta EE**
    - We'll still use the term Java EE in the course
  - **Java ME** – for smaller devices
    - Subsumed by Android for (non-Apple) smartphones

Notes:

- The standard Java Core API library is part of every Java SE installation. This includes thousands of classes from the `java` and `javax` packages – e.g., `java.lang`, `java.util`, `javax.naming`, `javax.xml.stream`, etc.
  - Over time, many "extension" packages, named "`javax.something`," were moved into the Core API library, but retained their original "`javax`" package name for compatibility.
    - As one example, the Swing UI library, originally `java.swing`, changed names to `javax.swing` and was pulled out as a "standard extension." Later, it was brought back into Core, but retained it's "`javax`" package name.
  - Don't get hung up on this – any package that's included in the "Java Platform, Standard Edition API Specification" (a.k.a. "Java Core API") is part of Core, regardless of its package name.
- Java ME is targeted for "smaller devices," before the huge popularity of smartphones.
  - Google's Android system is based on Java ME, but needed more, so it incorporated elements of Java SE, as well as its own vendor-provided libraries.
  - Java ME is still provided and maintained by Oracle.
- As of Java EE 8, this platform was transferred to the Eclipse foundation, and renamed Jakarta EE
  - See: https://www.eclipse.org/community/eclipse_newsletter/2019/august/jakartaee8.php

Lifecycle of a Java Program – Illustrated



Session 1: Review – Basics 20

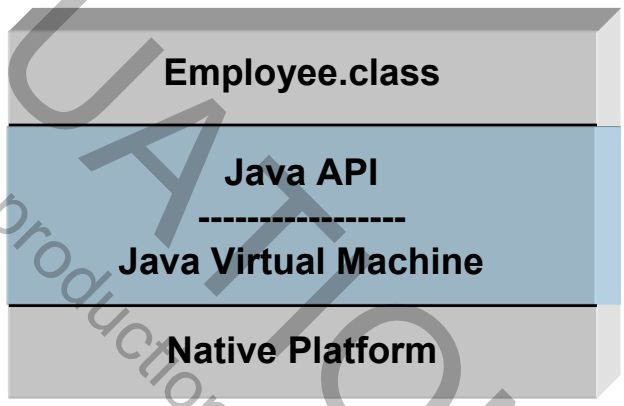
Notes:

- JDKs for Windows, Solaris, Linux, and macOS are available from Oracle. For an IBM z/OS mainframe or AIX system, it's from IBM, HP-UX from Hewlett Packard, etc.
  - Historically, the JVM for any given platform has generally been provided by that respective vendor. Note, however, that there are exceptions to this:
    - IBM provides a JDK for Windows (through Java 8). Its Rational Application Developer IDE bundles the IBM JDK for Windows, which is required to run RAD. IBM also bundles its JDK with the WebSphere Application Server on Windows, and requires it to run WebSphere.
  - Nowadays, you have a choice between the **Oracle JDK** and **OpenJDK**, on several platforms.
    - The open-source OpenJDK is an increasingly popular choice for today's main platforms: Windows, Solaris, Linux, macOS, and AIX. *http://adoptopenjdk.net*.

- Remember, the JDK and JVM are platform-specific, but the compiled bytecode (*.class* files) is portable across all JVMs.
  - For example, you can compile on a Windows JDK and run on a Linux JVM.

Java Language vs. Java Runtime

- Java platform executes bytecode, regardless of its source
  - Software-only platform comprised of the Java Core API and JVM
  - Source code can be written in other languages, too!
    - Groovy, Kotlin, Clojure, Scala, Jython, JRuby, others
    - These compilers create *.class* files – just like **javac**
    - These languages have full access to the Java API



Session 1: Review – Basics 21

Notes:

- The Java platform has become just as important as the Java language itself.
  - There is a huge win in this – you can write code in JVM languages like Groovy and Kotlin, taking advantage of each one's features, yet still retain access to the full Java API and the portability of the Java runtime environment.
  - From the JVM's perspective, a *.class* file is a *.class* file – bytecode is bytecode.

Applications, Deployments, and Libraries

- Application types:
  - **Standalone**: uses own JVM, bootstrapped via `main()` method
  - **Server-side**: runs in Java EE app server, uses app server JVM
    - Deployed into server as web app (WAR) or enterprise app (EAR)
- Packaging / deployment types:
  - **JAR**: standalone application or Java library
    - Contains *.class* files and resource files (1)
  - **WAR**: web application
    - Like a JAR, but has additional structure and Java EE XML files
  - **EAR**: generalized server-side application package
    - EAR = WAR(s) + JAR(s) together in one file
- Myriad 3rd party libraries also available – generally as JARs

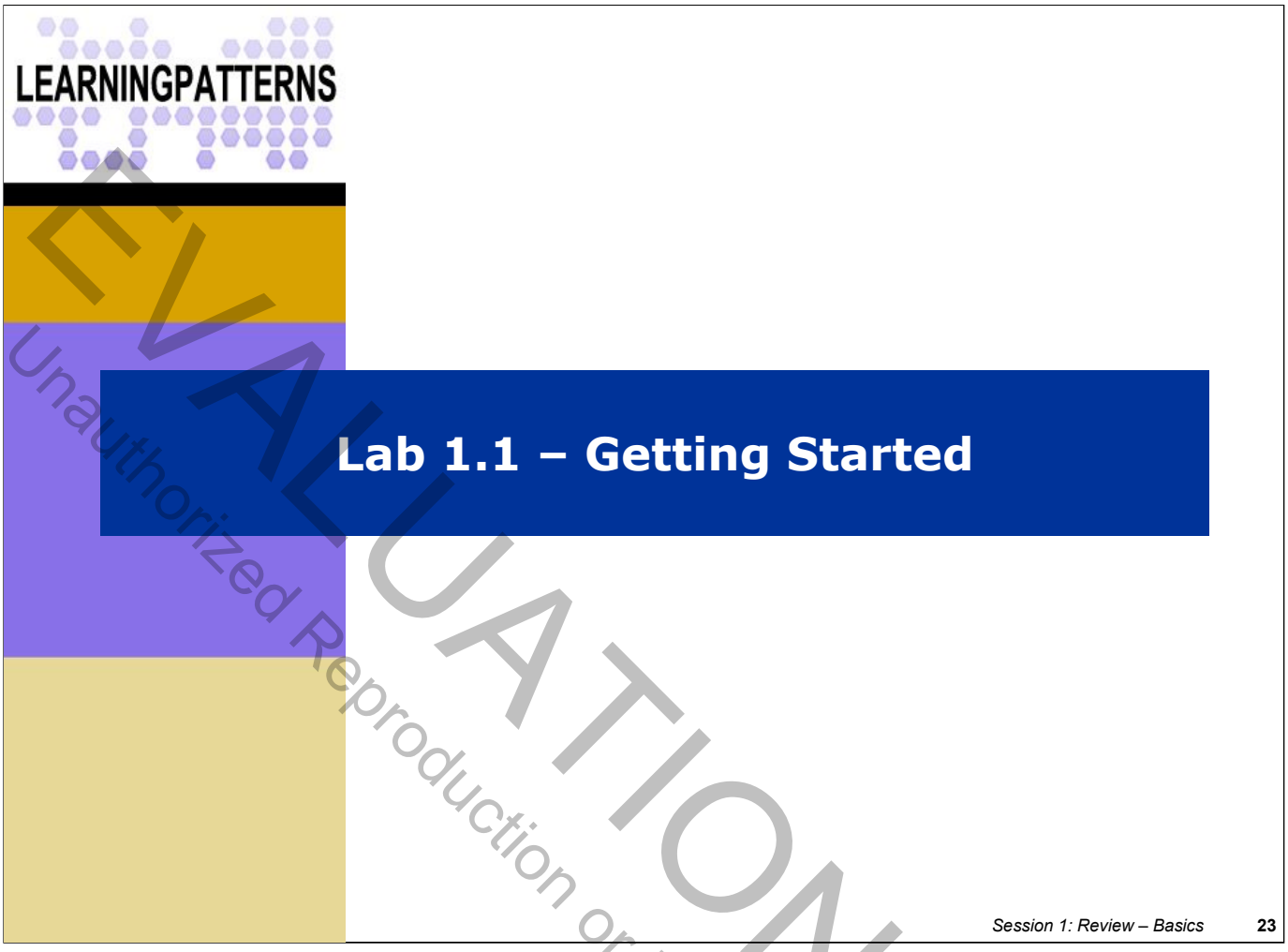
Session 1: Review – Basics 22

Notes:

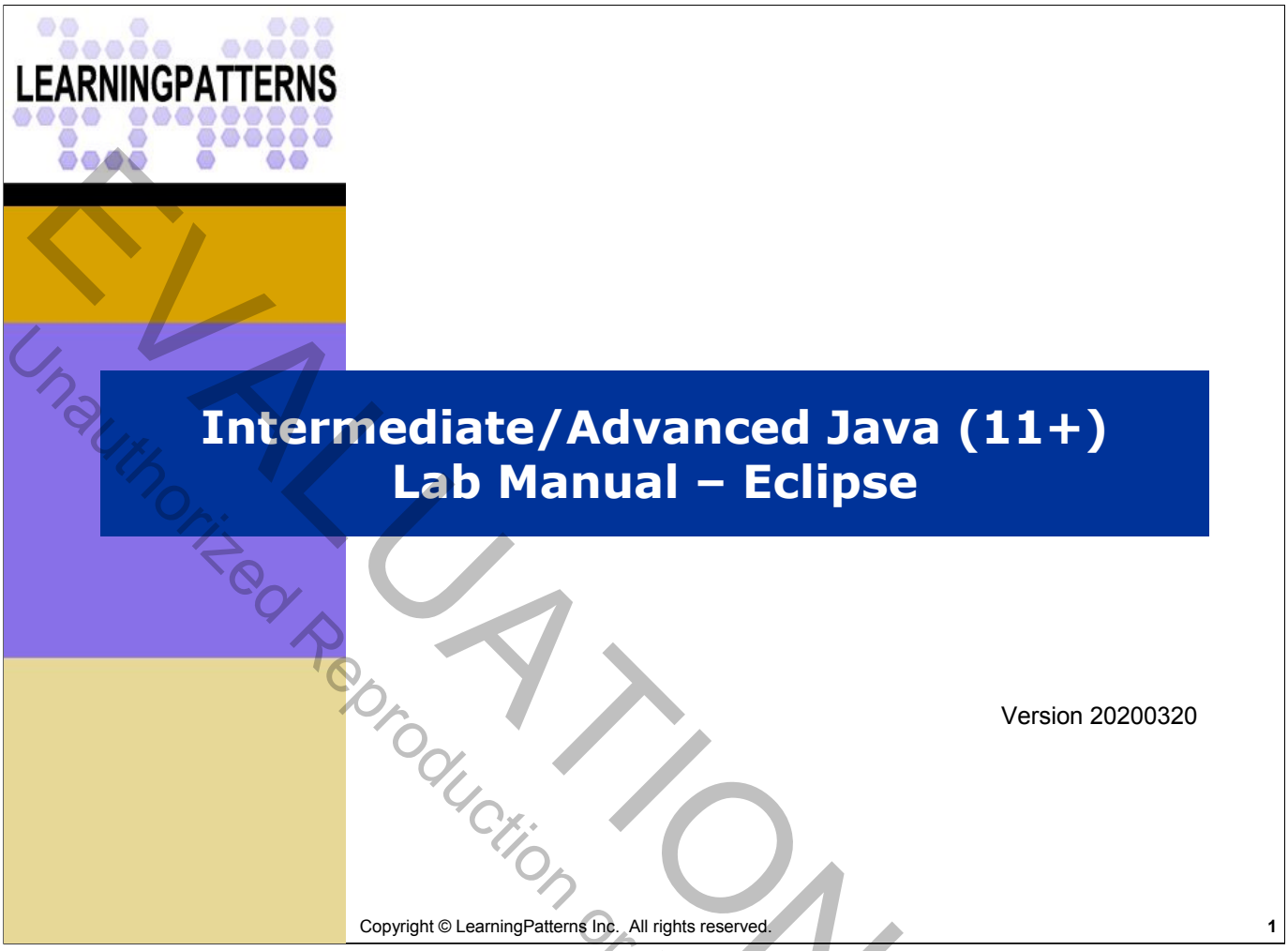
(1) Resource files are simply all files that are not *.class* files. They include *.properties* files, images, XML files, etc. – any files that might be needed at runtime.

- JAR / WAR / EAR files are just zip files with *.jar*, *.war*, and *.ear* extensions, respectively.
  - They are all created via the JDK **`jar`** utility or your IDE or build tool.
  - We go into the details of WARs and EARs in server-side development courses.

- There are countless 3rd party libraries available in Java, many of them open-source and free to use and distribute with your applications.
  - They run the gamut, from sophisticated enterprise libraries like Spring and Hibernate, to simpler utility libraries that allow you do such things as read an Excel spreadsheet from Java.
  - They are almost always packaged as JARs.



Notes:



Notes:

- Version 20200320

Release Level

Lab

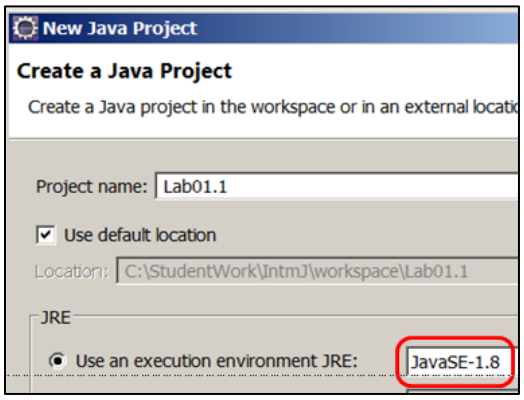
- This manual contains instructions for creating and running the labs using the following software packages:
  - **Java 11**
  - **Eclipse IDE for Enterprise Java Developers – 2019-12 or later**
    - Labs were developed and tested on **2019-12**
      Using this version or later is recommended
    - Version 2018-12 was the first release to fully support Java 11
- Instructions are geared towards Windows-based operating systems
  - You should easily be able to adapt to other environments, e.g., macOS
- **See notes** regarding screenshots

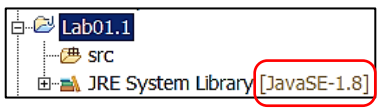
Introduction 2

Notes:

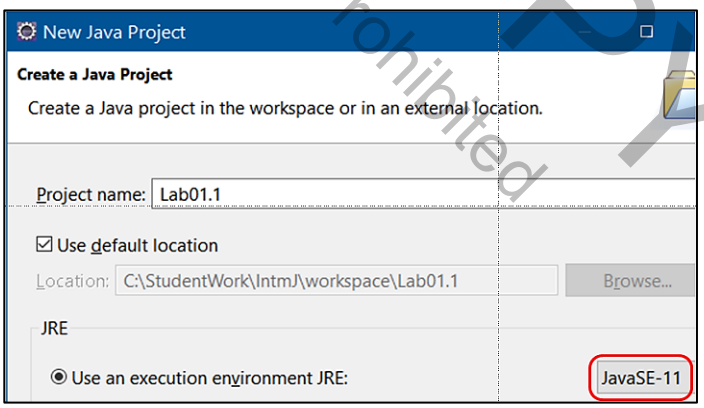
- The course was originally written for Java 8 and the labs developed with Eclipse Oxygen/4.7.
  - We have not updated every screenshot, because in many cases the differences are minor and not important to completing the lab.

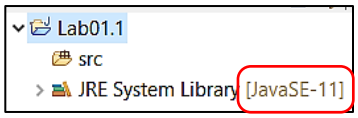
Eclipse 4.7 Classic Theme with Java 8
Windows 7 Classic Theme

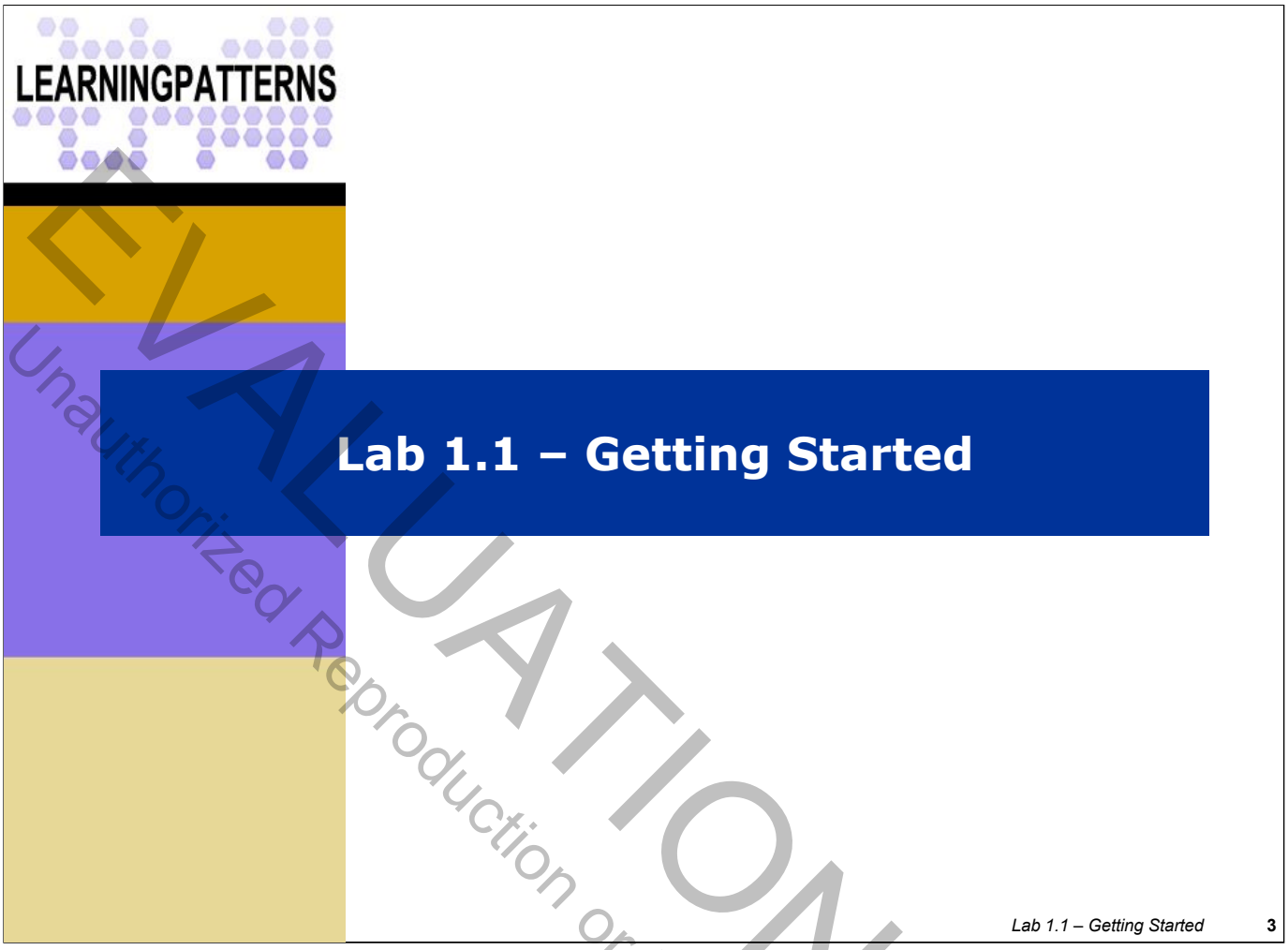




Eclipse 2019-12 Classic Theme with Java 11
Windows 10 Aero Theme







Notes:

Lab Synopsis

- **Overview**:
  - Set up our lab environment and the Eclipse IDE
  - Write the canonical HelloWorld application in the IDE, and run it there
  - Run the application from the command line
  - Create a JAR file for our application and run it using the JAR
  - Extract and bookmark the Java Core API Javadoc
- **Builds on previous labs**: none
- **Approximate time**: 15-20 minutes

Lab 1.1 – Getting Started 4

Notes:

Information Content and Task Content

Lab

- **Informational content** is presented in the normal way – the same as in the student manual
  - Like these bullets at the top of the page
- **Tasks** you need to do are in a "Tasks to Perform" box

Tasks to Perform

- Look at these instructions, and notice the different look of the box
  - Make a note of how it looks, as future labs will use this format
- Make sure that you have **Java 11 installed**
  - Likely in a directory such as ***C:\Program Files\Java\jdk-11.0.6*** (1)
  - If not, you'll need to install it – it can be downloaded from: ***http://oracle.com/technetwork/java/javase/downloads***
- OK – now **get out your setup files**, we're ready to start working

Notes:

(1) On macOS, the JDK is typically installed under */Library/Java/JavaVirtualMachines/jdk-11.0.6*.

- The setup files may be on your workstation, or they may be provided by your instructor.

Extract the Lab Setup Zip File

Lab

- To set up the labs, you'll need the course setup zip file
  - It has a name like: ***LabSetup_IntmJ_20200330.zip***
- Our base working directory for this course will be ***C:\StudentWork\IntmJ***
  - Gets created when we extract the lab setup zip
  - Includes a directory structure and files needed in the labs
  - All instructions assume that this zip file is extracted to ***C:***

    If you choose a different directory, please adjust accordingly

Tasks to Perform

- Unzip the lab setup file to ***C:***
  - Creates the ***StudentWork/IntmJ*** directory structure, containing files that you will need for doing the labs

Lab 1.1 – Getting Started 6

Notes:

- The setup files may be on your workstation, or they may be provided by your instructor.

Java Core API

Lab

- There are **thousands** of classes in the Java Core API
  - It's **critical** that you know how to read their APIs and use them
- Available online, but nice to have it "installed" as part of your JDK
  - Online: ***http://docs.oracle.com/en/java/javase/11/docs/api***
  - Installed: download docs zip and extract into JDK installation directory
    - **Note**: separate download from the JDK installer executable

Tasks to Perform

- On the filesystem, find *C:\StudentWork\IntmJ**Resources***
  - Note the file ***jdk-11-api-docs.zip*** (exact filename may be different)
  - Extract this zip file to your Java installation directory – likely in a directory such as ***C:\Program Files\Java\jdk-11.0.6*** (1)
- Open ***<jdk>/docs/api/index.html*** in a browser – **and bookmark it**
  - See notes regarding differences from previous versions of Javadoc (2)

Lab 1.1 – Getting Started 7

Notes:

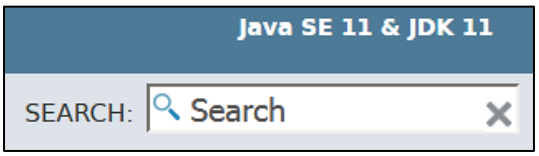
- It's nice to have the documentation installed locally, so you're not dependent on an Internet connection. If you're using a later update than 11.0.6, these docs still apply.
- We'll sometimes use ***<jdk>*** as an abbreviation for the JDK installation directory.

(1) On a Windows system, you may not be able to extract the zip directly into the Java installation directory, because it's under the special "*Program Files*" directory.

- As a workaround, you can extract to a temp directory, or do an "extract to here," then **move** (don't copy) the resulting ***docs*** directory to your JDK installation directory.

(2) Looks familiar, but the front page is a list of **modules** now, not packages.

- To get to the packages, click on module **`java.base`** and then it's business as usual.
- Note the **Search** field upper right!



The Eclipse Platform

Lab

- **Eclipse** (**www.eclipse.org**) is an open source platform for building integrated development environments (IDEs)
  - Used mainly for Java development – can be extended via plugins and used in other areas (e.g., C# programming)
- Originally developed by IBM
  - Released into open source
  - IBM's RAD product line is built on top of Eclipse
- Eclipse products have two fundamental layers:
  - **Workspace**: files, packages, projects, resource connections, configuration properties – all stored on the **filesystem**
  - **Workbench**: editors, views, and perspectives – the **UI** you work with
- We will set up the workspace and workbench, then do our lab

Lab 1.1 – Getting Started 8

Notes:

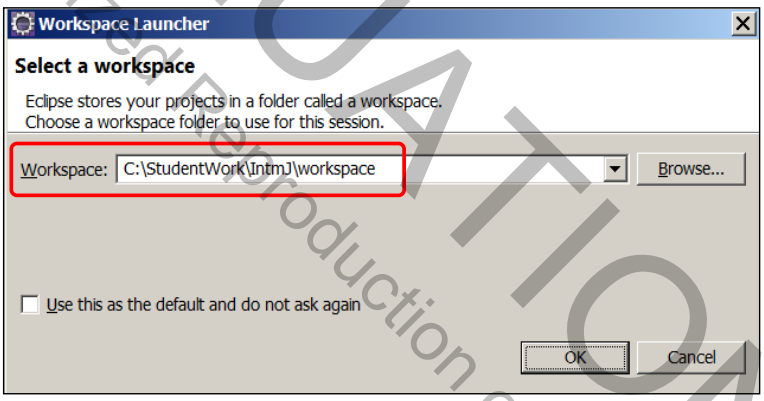
- The workbench sits on top of the workspace and provides visual artifacts (the UI) that allow you to access and manipulate the underlying workspace resources.

Getting Started with Eclipse

Lab

Tasks to Perform

- Make sure you have **Eclipse installed** – likely in ***C:\eclipse***
  - If not, you'll need to install it – see instructions in notes
- **Launch Eclipse**: go to ***C:\eclipse*** and run ***eclipse.exe***
  - A dialog box should appear prompting for a workspace location (1)
  - Set the workspace location to *C:\StudentWork\IntmJ**workspace*** (2)
  - If a different default workspace location is shown, change it
  - **Don't import any projects** – just open the workspace (3)



Lab 1.1 – Getting Started 9

Notes:

(1) If the workspace dialog box does **not** appear, that means that Eclipse was previously launched on your system, a workspace directory was chosen, and the "Use this as the default..." box was checked.

- In this case, you should change workspace to that shown above via:

  File → Switch Workspace → Other... This will open the dialog box.

(2) **NOTE: you must use our provided workspace directory**, as we've supplied starter code and other files for some of the labs here.

(3) You should not do anything but open Eclipse using the correct workspace here.

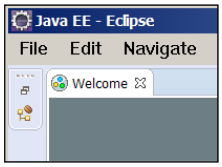
- In particular, you should not import any project folders you see. We'll explain how to use these all correctly later, and it's important to wait until then.

- If Eclipse was installed elsewhere, adjust the path to the Eclipse executable accordingly.
- You can also put a shortcut on your desktop to start Eclipse.
- If you need to download Eclipse, go to *http://www.eclipse.org/downloads/packages*.
  - **Ignore** the large banner "Try the Eclipse Installer" and instead use the links under the heading "Eclipse IDE for Enterprise Java Developers." These links are for the zip file installations.
  - Save the zip file to your computer, and unzip it. The easiest location to unzip it to is *C:*, but another location is fine as long as you can get to it to run the *eclipse.exe* executable.

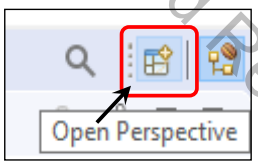
Workbench and Java Perspective

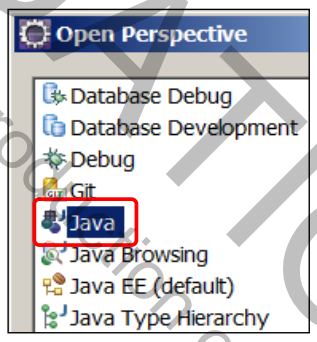
Lab

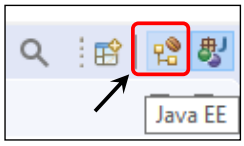
Tasks to Perform



- Close the Welcome screen (click the X on its tab)
- **Open a Java Perspective**
  - Click the Perspective icon at the top right of the Workbench and select **Java** (see below left and below center)
  - See notes about perspective icons
- **Close the Java EE perspective** by right-clicking its icon at the top right of the Workbench and selecting Close (see below right)
  - The Java EE perspective is the default for the Eclipse Java EE version







Lab 1.1 – Getting Started 10

Notes:

- **NOTE**: the perspective switcher does not show the perspective name by default. To change this, right-click on any perspective icon and **Show Text** – see examples below.

without labels

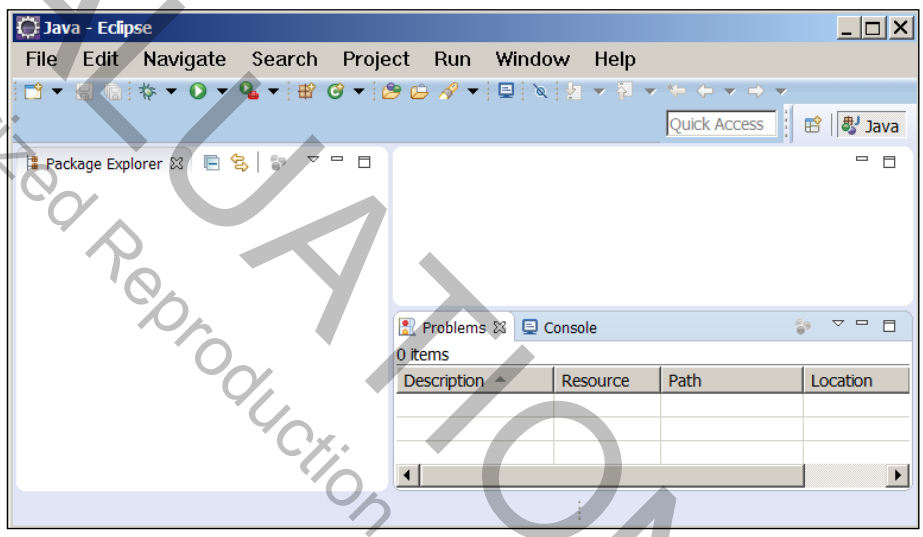
with labels

Customize the Perspective

Lab

Tasks to Perform

- Unclutter the perspective by closing some views
  - Close the Task List and Outline views (click on the X)
- Open the Console view: **Window → Show View → Console**
- You can save these changes to the perspective (see notes)

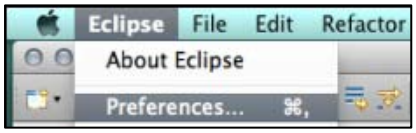


Notes:

- An Eclipse ***perspective*** is simply a collection of ***views***, shown as tabbed panes in the Workbench.
  - A perspective defines which views are included, where they are positioned, and what their relative sizes are in the Workbench.

- Eclipse has several predefined perspectives, which can be used as-is, or customized to suit your needs / preferences. There's a good chance you'll want to tweak the ones you're using.
  - To customize a perspective, open a predefined one, e.g., the Java perspective, close the views you don't want, open any views you do want, reposition and/or resize them, and then save the changes via **Window → Perspective → Save Perspective As...**
    - TIP: to preserve the factory-shipped perspective and create a brand new perspective based on it, use a different name, e.g., "Java [YI]", where "[YI]" are Your Initials in square brackets.
    - Window → Perspective → Save Perspective As... → Java [YI].

- Your instructor may also recommend additonal customizations, which s/he will go over with you.
  - There are myriad settings, all available via **Window → Preferences**.
  - On macOS, the Preferences panel is reached via **Eclipse → Preferences**.

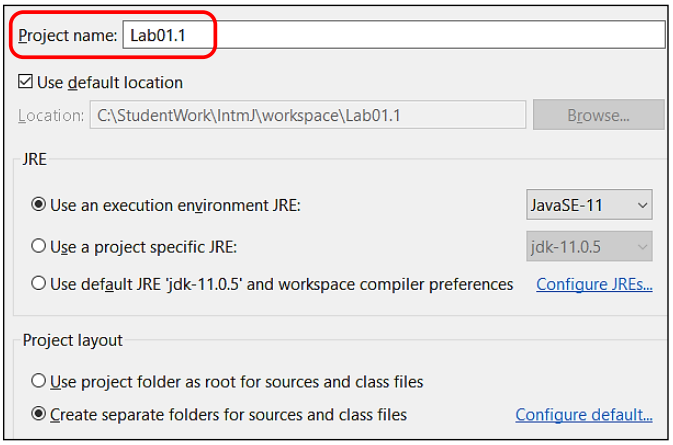


Create Project

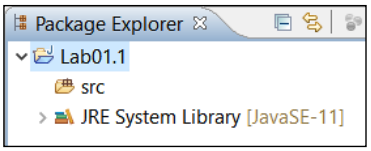
Lab

Tasks to Perform

- Create a new **Java** project
  - **File → New → Java Project**
  - Name it **Lab01.1**
- Note the remaining defaults
  - Location: ***workspace/Lab01.1***
  - Separate folders for source and class files (best practice)



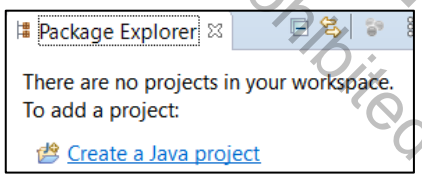
- **Finish**
  - When prompted for module name, click **Don't Create**
    - **NOTE: always** choose Don't Create until told otherwise – more on this later *
- Your new project in the Package Explorer view:



Lab 1.1 – Getting Started 12

Notes:

- There are multiple ways to create a new project.
  - File → New → ...
  - Ctrl+N.
  - Click on the "New" wizard icon on left side of the toolbar.
  - Right click anywhere in the Package Explorer view, select New ...
  - Use the link "Create a Java project" in the Package Explorer view.
    - Note that this link is only present when you have no projects yet.



- It's a best practice to create separate folders for source and class files.
  - Eclipse puts the *.java* files in the lab's *src* directory, and the *.class* files in the lab's *bin* directory.
  - This is the default structure for Java projects in Eclipse.

* We'll work with the new Java module system later.
  - For now, just **never** create a module for your projects.

Write and Run HelloWorld

Lab

Tasks to Perform

- Create package named **com.hello**
  - Right-click on ***src*** and choose **New → Package**
  - All classes should be in a package, even this simple one
- Create class **HelloWorld** in the com.hello package
  - Right-click on package name and choose **New → Class**
  - Complete the class as shown

```
package com.hello;

public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello world!");
  }
}
```

- Run it: right-click on ***HelloWorld.java*** and choose **Run As → Java Application**

Lab 1.1 – Getting Started 13

Notes:

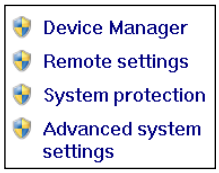
Set JAVA_HOME

Lab

- To use the JDK tools on the operating system command line, you should set the **JAVA_HOME** environment variable
  - Then add ***JAVA_HOME/bin*** to your **PATH**
  - **May already be done** – check for it first

Tasks to Perform

- Find where your JDK is installed
  - Likely in a directory such as ***C:\Program Files\Java\jdk-11.0.6***
- Open the Windows **System** Control Panel
  - Choose "Advanced system settings"
  - In the dialog, select "Advanced" tab and click "Environment Variables" button
    - Create or modify the **JAVA_HOME** variable to point to your JDK installation
    - Add **%JAVA_HOME%\bin** to the **front** of the **PATH** variable
  - See your instructor if you need help



Lab 1.1 – Getting Started 14

Notes:

- Setting JAVA_HOME may have been specified in your Classroom Setup instructions. It's always a good idea to have it, even if you do "everything" from the IDE, because at some point you'll need to run something or use one of the JDK tools on the command line.

Run the Application Externally

Lab

- You'll use the `java` command line tool to run the app
  - Will need to specify the ***classpath*** – where `java` looks for classes
  - Best practice: use the `-classpath` flag (see notes for details)

```
$ java -classpath <path-to-classes;path-to-jars> YourClassName
```

Tasks to Perform

- Open a command prompt to ***C:\StudentWork\IntmJ\workspace\Lab01.1***
  - The *HelloWorld.class* file is in a package directory under *Lab01.1/**bin***
- Run `HelloWorld` from the prompt – use **fully qualified classname**

```
C:\StudentWork\...\Lab01.1>java -classpath bin com.hello.HelloWorld
Hello world!
```

  - If you get a "command not found" error, there's a problem with your `JAVA_HOME` and/or `PATH`

Lab 1.1 – Getting Started 15

Notes:

- The `java` command line tool is the JVM launcher.
  - It requires a main-class to run (a class with a `main()` method).

    ```
    $ java YourClassWithMain
    ```

  - For a usage banner, just type `java` at the command prompt and press [Enter].

- We're using a relative path for the classpath (relative to *Lab01.1*) – the ***bin*** directory.
  - That directory contains our "loose" *.class* file – *HelloWorld.class*, physically stored in the appropriate package subdirectory *com/hello*.
  - The runtime classpath can be set two ways:
    - `CLASSPATH` environment variable (global to the operating system).
    - The `-classpath` flag specified at runtime (which overrides any `CLASSPATH` setting).
  - If the `CLASSPATH` environment variable is not set, and the `-classpath` flag is not provided, the classpath defaults to the current directory (`.`).

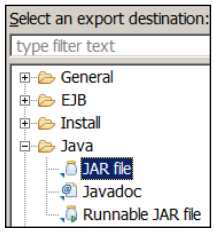
Create Application JAR File

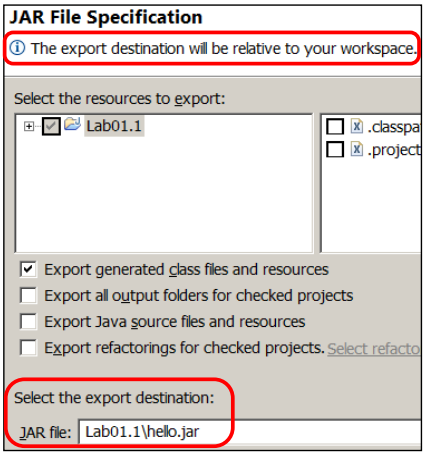
Lab

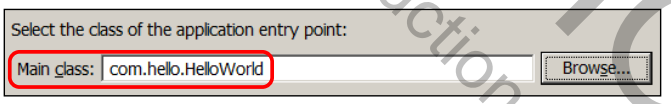
- Java software is generally shipped in JAR files
  - We'll create one here, and use it to run our application

Tasks to Perform

- In Eclipse, right-click on ***src*** and choose **Export...**
  - Expand the "Java" category and choose "JAR file"
  - Next
- In the next dialog, specify destination
  - Easiest to just type it in: ***Lab01.1\hello.jar***
    - File is created relative to *workspace* directory
  - Press Next **twice**, to the dialog named "JAR Manifest Specification" (see notes)
- Specify that the JAR has a main-class
  - **Finish**

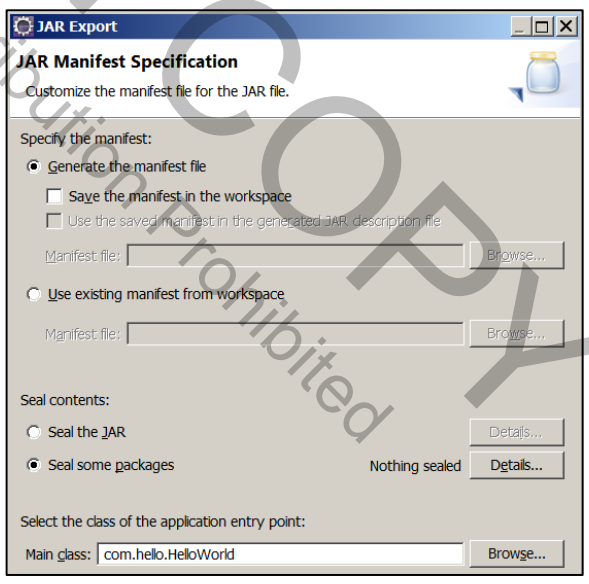






Notes:

- Full dialog screenshot at right.



- Details on the main-class specification:
  - Every JAR file has a ***MANIFEST.MF*** file, stored in the ***META-INF*** directory of the JAR.
    - It's a simple text file with name-value pairs.
    - Various things can be specified, using standardized manifest file attributes.
  - We're specifying that the JAR has a main-class, and we're providing its name.
  - The resulting manifest file will look like this:
    `Manifest-Version: 1.0`
    `Main-Class: com.hello.HelloWorld`
  - This allows us to "run the JAR" as an application. We do this on the next page.

Run the Application Using the JAR

Lab

Tasks to Perform

- From your command prompt in ***workspace/Lab01.1***:

```
$ java -classpath hello.jar com.hello.HelloWorld
Hello world!
```

  - Here, `-classpath` specifies the location of the JAR file
  - As before, we're using a relative path (relative to *Lab01.1*), and we need to specify the fully qualified name of the main-class

- Now run it as an "executable JAR" via the `-jar` flag

```
$ java -jar hello.jar
Hello world!
```

  - Relies on the JAR having a `Main-Class` attribute in its manifest file
    - We specified this during JAR file creation
    - That's why you don't have to specify the classname of the main-class when you run it

Notes:

Create a Launch Script [Optional]

Lab

- Many applications ship with a launch script
  - End user simply executes the script, which sets up the classpath, and may additionally configure other application-specific properties

Tasks to Perform

- Using a text editor, create a ***run-hello.cmd*** file in the *Lab01.1* directory
  - Here's a very simple one:

```
@echo off
rem     Executes the HelloWorld application
java -jar hello.jar
```

- Try it out:

```
$ run-hello
Hello world!
```

Lab 1.1 – Getting Started 18

Notes:

Lab Summary

- This "simplest of all labs" still contained quite a bit of work(!)
- In this course, we are exposing you to some of the things you **need to know** as a more experienced Java developer – in this case:
  - JAVA_HOME
  - Classpath
  - Creating and using JAR files
  - Application launch scripts
- These are things you will repeatedly encounter

Lab 1.1 – Getting Started 19

Notes:

7400 E. Orchard Road, Suite 1450 N
Greenwood Village, Colorado 80111
Ph: 303-302-5280
www.ITCourseware.com